Bachelor's Degree Final Thesis

B. Tech. in Industrial Technology Engineering

A Prediction Model for Neuronal Synaptic Inputs

MANUSCRIPT

June 21, 2020

Author: Pau Fisco Compte

Supervisors: Enric Fossas Colet
Nestor Roqueiro

Convocation: 02/2020

ETSEIB

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

UPC

Abstract

In this manuscript it is exposed a method to approximate functions using artificial neural networks based on wavelets (wavenet). The focus is on finding the best configuration for the wavenet, from various possible settings in relation to the mathematical development of the network, to be able to approximate a FitzHugh-Nagumo model to later be used to predict any scenario with a single initial condition for the model. It is shown that after training the artificial network, it is able to approximate the non-linear behaviour of the FitzHugh-Nagumo model with high accuracy, providing a neuron model which can be then applied to models for real neurons with similar inputs to those from the wavenet. Finally, it is proved that additional linear terms applied to the outputs improve significantly the error of the approximation to those wavenets with non-linear type scale functions, thus it is been possible to obtain better results without the need to increase the resolution level and thereby reducing the simulation time.

Contents

ETSEIB

1 Preface

Computational neuroscience is the theoretical study of the brain to uncover the principles and mechanisms that guide the development, organization, information processing, and cognitive abilities of the nervous system [1]. The general paradigm in computational neuroscience is focused in the study of neurons, and using that knowledge one can design an application that can solve specific problems according to the functionality of these cells. For example, a grasshopper neuron that processes images [2], for any computer vision applications. This scenario generates the necessity to study different neurons, probably from different kinds of simple animals, to create any desired application.

In engineering, all those specific neurons are pretended to be obviated and the goal is to develop one model or few, to create generic algorithms for any kind of applications.

1.1 Project's origin

This manuscript surges from a previous work [3], in which the Morris-Lecar (ML) model is used to describe the behaviour of a neuron. The model's goal is to create data for a neural network (NN) based on wavelets, a wavenet, without the need to know the intrinsic properties of the neuron, its morphology, the ionic channels, the neurotransmitter availability, etc.

The challenge of the present project, as in the referred previous work, consists on being able to predict the behaviour of an excitable system, like one described by a neuron, considering it as a black box, thanks to the use of a wavenet, using the FitzHugh-Nagumo (FHN) model instead of the ML worked on [3].

1.2 Motivation

The growing interest in artificial neural networks and the control over Python have been a motivation to try to reproduce similar work done in [3]. If the inputs of a neuron can be predicted using a NN, it is possible to predict more complex neurons making variations of the model with the same kind of wavenets. With this work it would be possible to create a wavenet that can be used for complex industrial processes [4], because these can be difficult to describe due to the amount of variables interacting.

One of the jobs of an Industrial engineer is to optimize processes, and one way to optimize nonlinear complex systems is with the use of NNs and artificial intelligence. Many applications have focused in this direction and this thesis is an introduction to what the future has to offer.

1.3 Previous requirements

To be able to make this project, it is necessary to have basic knowledge about Python3, due that the wavenet will be written in this programming language. It is necessary to know how to implement certain scientific libraries as *numpy* and graphic libraries such as *matplotlib*.

It is also necessary to have a background about ordinary differential equations (odes), as this kind of equations will be the ones conforming the mathematical models, describing the neuron behaviour (like the ML and FHN models), to be able to train the wavenet. The Euler's method will be used to solve these equations, to compare the analytical solution of the functions with the predicted values from the NN.

2 Introduction

This document diverges from the tools used in previous works in which this project is based. To create the NN, Python is directly used from the equations behind the wavelet's theory. These equations have been developed and adapted to satisfy the needs of the wavenet. It is created without specific tools for this purpose or any tool used in NNs, like the ones offered in *Tensorflow* or *Pytorch*. All wavenet's routines have been integrally developed. For this reason, the optimization of the code also has been an objective, so the training time can be reduced and changes in the wavelet's configuration can be applied easily.

Starting from the work accomplished in [3], there is the possibility to obtain real data from directly measured voltages from the neuron of a giant barnacle (following the ML model) and to test the behaviour of the wavenet in a real world scenario. As seen in [3], the model works successfully, and in this project it is presented the same goal but with the FHN model.

First of all, it is important to get used to wavelets and the different types of NNs, to know which are the ones that fit the purpose of the project. Once implemented these equations to the NN, it will get more complex by increasing the number of inputs and adding linear terms and an offset, if it is necessary.

2.1 Project objectives

The principal objective of this project is to create a wavenet, trained with a grid of initial conditions, large enough as to contain all possible outputs of the FHN model in front of any random stimulation. The objective can be divided in more simple ones:

- To understand the concepts about the theory of wavelets and how to use them to predict functions.
- To know the different topologies of a NN, and the one fitting a wavenet, that is the number of neurons and the relations among them.
- To find a way to obtain the predicted function in a fast and precise form, making use of the theory around wavelets and any possible Python programming technique, to acknowledge the best structure configuration for the wavenet, focusing on the goal to obtain a minimum training time and a minimum error.

2.2 Points included

With a three months investigation and development of the NN using Python, the phases of this document are:

- To search for the possible structures for the wavenet and the different resolution methods to find the predicted function.
- To develop algorithms that satisfy the correct use of neurons for the type of problem.
- To study ways to optimize the Python code to accelerate the training by making it faster and by reducing the amount of calculations needed to obtain a wavenet with less complexity with the same accurate result.

- To do an economic analysis to study the project's cost and its environmental impact.

3 Approach to neural networks

Before going deep in details about the wavenet developed in this project and explaining how it has been obtained through the use of wavelets, it is necessary to understand the general structure of NNs.

NNs are a powerful machine learning architecture to approximate arbitrary input-output functions once given enough training data. As the name says, neural networks are inspired in the brain and its neurons. The neuron in a NN is the basic element, serving as a node in the different layers of the network.

As it is shown in figure 3-1, there are some input signals x_i which go to the node, that does some mathematical calculations on those inputs and gives a y output.

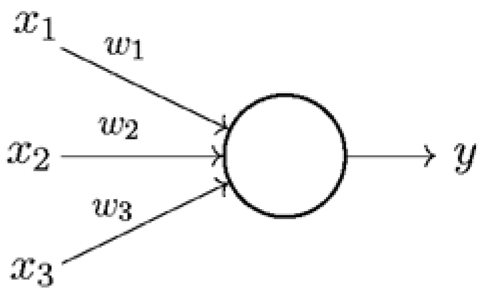

Figure 3-1. Structure of a neuron's computational model.

This mathematical calculation executed in a neuron is done by an activation function, and the sigmoid is frequently used as such. If the input in a sigmoid is small the output will be near to zero. If the input is large, regardless of how large it is, the output will be near to one. As a note, the hyperbolic tangent can be used as a sigmoid because it has nearly the same behaviour.

Nowadays, it has increased the use of an activation function called ReLU, where the function takes zero value until a point and then grows linearly with u, where u is the value taken by the input of the neuron.

There is assigned a weight w_i to each one of the connections between layers, from a neuron to the next one. These weights are just numbers that act as coefficients, once the values from the neurons of the previous layer computed their weighted sum according to these weights.

Is not needed to use an activation function in every single neuron. For instance, the output layer can be different linear relations from the previous layers using the values of each neuron, and computing their weighted sum according to the weights between the last two layers of the NN.

If the neurons are stacked together creating layers, the structure of the layers plus the different activation functions used in each neuron, they can become a different kind of NN with interesting properties for each application. As seen in figure 3-2, the stacking of the neurons can create the artificial neural network (ANN), in which there is one input and one output layer, and any layer of neurons in between is called a hidden layer.

The different nodes can have different activation functions, depending on the structure of the NN the different nodes can have just linear relations to other ones or one node can have an offset, that can be called bias as well, to add a constant value to the relation.

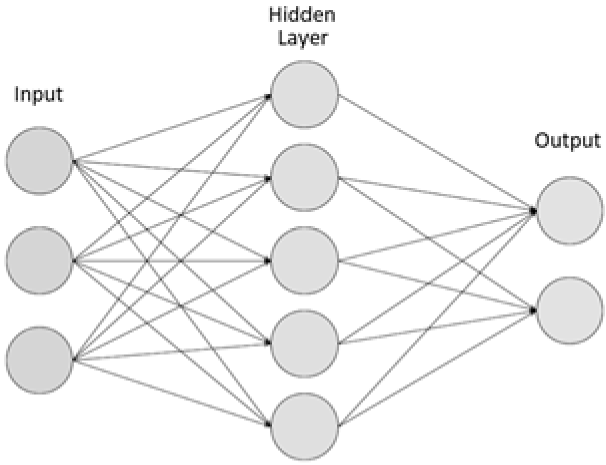

Figure 3-2. Structure of a simple feedforward NN.

If the NN has more hidden layers added next to each one, so each one of them are doing some kind of sequential processing it is called a deep neural network, and it becomes the basis of the deep learning as it is known today.

There is a massive number of topologies for NN, and most of them are described by the Asimov Institute [5]. For example, convolutional neural networks are good for image recognition and long/short-term memory network are good for speech recognition.

Finally, it is important to mention that there are many ways to optimize the learning of a NN [6], given by the optimal value of the parameters involved in the NN. The optimization is computed using the loss function, by matching the target value from the dataset used, and the predicted value obtained from the output of the NN. Later in the project, it will be discussed which is the loss function used in the wavenet and how it is used.

4 Mallat's multiresolution analysis

Before talking about wavenets, it has to be mentioned the multiresolution analysis developed by Mallat, so the use of the wavelets' equations, and the wavenet can be understood more deeply.

Mallat developed this theory around the Hilbert space, a particular case of a metric space composed by Lebesgue integrable functions L^n, where the dot product of two functions is given by the expression in (4.1). Hilbert defined a L^2 space with a dot product of a function and the transposed of another in a compact domain, as a result it can be defined an orthogonal family of those functions.

$$\langle f, g \rangle = \int_{-\infty}^{\infty} f(x)g(x)dx \tag{4.1}$$

As described in [7], Mallat took the wavelets and scale functions and brought them to the Hilbert space $L^2(\mathbb{R})$, thus it is possible to create an orthogonal base that can define any expression as a sum of linear combination of wavelets and scale functions. Here is born the multiresolution analysis, a technique used to approximate a function at finer scales of resolution, as explained in [7, 8, 9].

Defining a multiresolution analysis in $L^2(\mathbb{R})$, the sequence of V_j subspaces, where $j \in \mathbb{Z}$, have to satisfy the next conditions described in [7, 8, 9].

• Nesting property

Defined by the expression in (4.2), where a function $\phi(x)$ belongs to V_0 and a set of its translations $\phi(x-n)$, where $n \in \mathbb{Z}$, represents an orthonormal base in the same subspace.

$$...V_{-2} \subset V_{-1} \subset V_0 \subset V_1 \subset V_2... \tag{4.2}$$

The $\phi(x)$ function is called scale functions and in a *j* resolution it is orthonormal only to those functions from the same resolution level.

• Scaling and orthogonality

If $\phi(x) \in V_0$, then $\phi(x) \in V_1$. Given by the expression (4.3), V_1 is ruled by $\phi(2x)$ functions, and $\phi(x)$ can be expressed as a combination of $\phi(2x)$ functions, so $\phi(x) \in V_1$ also.

$$f(x) \in V_j \iff f(2x) \in V_{j+1} \tag{4.3}$$

• Density and separation

Any $f(x)$ function that can be expressed in a Hilbert space, can be defined in a j resolution as the projection of the function through the V_j subspace. But this projection may not have all the information from the $f(x)$ function. When the resolution grows tending to infinity, the approximated function converges to the original function $f(x)$. This behaviour is defined by the density, as seen in the expression (4.4).

$$\overline{\cup_{j\in\mathbb{Z}} V_j} = L^2(\mathbb{R}) \tag{4.4}$$

On the other hand, when the resolution is decreased tending to minus infinity, the approximated function converges to zero. This behaviour is defined by the separation, as seen in the expression (4.5).

$$\cap_{j\in Z} V_j = \{0\} \tag{4.5}$$

To sum up, the projection of any function to an orthonormal scale functions' base has most of the information of the projected function, to define the rest of the function it is also projected to orthogonal basis, and these are wavelets' basis. So the original function can be expressed as a sum of linear combinations of scale functions from different resolution levels, whose at the same time, they can be expressed as a sum of linear combinations of orthogonal wavelets from upper resolution levels.

Having that said, the general expression of a predicted function that represents what Mallat defined, can be seen in (4.6).

$$f(x) = \sum_{n=-\infty}^{N=\infty} d_n \phi_n(x) + \sum_{m=0}^{M=\infty} \sum_{n=-\infty}^{N=\infty} c_{m,n} \psi_{m,n}(x) \tag{4.6}$$

Where:

$c_{m,n}$: n-th weight in the m-th level of resolution.

d_n: n-th weight in the first level made of scale functions.

ϕ_n: n-th scale function from the first level of resolution.

$\psi_{n,m}$: n-th wavelet from the wavelet family in the m-th level of resolution.

5 Approach to wavelets

Once understood the multiresolution analysis, it will be easier to comprehend the following wavelet expressions. Wavelets are a good mathematical tool to obtain frequency and time information from signals. They are finite waves with dilatation and translation parameters which can provide information about the width and high of the studied signal thanks to dilatation, and the location in time of the studied signal's wave thanks to translation.

From these dilatations and translations, it is given a family of functions called wavelet family. The basic function of the family is called mother wavelet ψ, and this wavelet family is expressed with the equation (5.1).

$$\psi_{a,b}(x) = |a|^{-1/2}\,\psi(\frac{x-b}{a}) \qquad a, b \in \mathbb{R}; a \neq 0 \tag{5.1}$$

Parameter b of the mother wavelet corresponds to translation while parameter a corresponds to dilatation. This way a temporal function can be explored with greater frequency resolution.

The value of a and b in the equation (5.1) can be defined in a discrete set, where $a = a_0^{-m}$ and $b = nb_0 a_0^{-m}$. Being $a_0 < 1$ and $b_0 > 0$ the wavelet family can be defined as the equation (5.2). For this project it is used the coefficient $a_0 = 2$ and $b_0 = 1$, as seen in [10].

$$\psi_{m,n}(x) = |a_0|^{m/2}\,\psi(a_0^m x - nb_0) \qquad m, n \in \mathbb{Z} \tag{5.2}$$

A mother wavelet is obtained from scale functions, as described in [8, 10] a scale function fulfils the property shown in the equation (5.3).

$$\phi(x) = \sum_{n=0}^{N} p_n \phi(2x - n) \tag{5.3}$$

Where p_n are the interscale coefficients. From this expression a mother wavelet $\psi(x)$, developed in the orthogonal base of the scale function $\phi(2x - n)$, can be expressed as shown in (5.4).

$$\psi(x) = \sum_{n=0}^{N} (-1)^n p_n \phi(2x - n) \tag{5.4}$$

Applying the orthogonality rule in (5.3), seen in (4.2), it becomes the known wavelet expression (5.4), that is, if $\phi(x) \in V_0$ then $\psi(x) \in V_1$. Defining the W_i subspace as the orthogonal complement of V_j, in the V_{j+1} subspace, if $\psi(x) \in V_1$, $\psi(x) \in W_0$, so $W_0 \subset V_1$, and wavelets also satisfy the same properties as the scale functions. The scale functions are orthogonal to a wavelet from an upper resolution level, whereas any wavelet is orthogonal to all wavelets from any resolution level.

The expression (5.4) has to satisfy the condition $\sum_{n=0}^{N} p_n = 2$, as seen in [10]. With this condition is possible to elaborate a table with the interscale coefficients of the previous sum, for the scale functions used in this project, as is seen in the table 4.1.

ETSEIB

Table 4.1 – Coefficients for wavelets generation according to their scale functions.

Scale function	Haar	Hat	Quadratic spline	Bicubic spline
Interscale coefficients	P0 = 1 P1 = 1	P0 = 1/2 P1 = 1 P2 = 1/2	P0 = 1/4 P1 = 3/4 P2 = 3/4 P3 = 1/4	P0 = 1/8 P1 = 1/2 P2 = 3/4 P3 = 1/2 P4 = 1/8

Wavelets can be used in a huge range of fields, for example, in non-stationary signal analysis, electrocardiogram analysis, seisms, sound, radars or even in image processing. In every case there will be one or few wavelets with specific properties related to every type of analysis. The three scale functions seen in the table above, that correspond each one to a different wavelet, are chosen for reasons that will be discussed in later chapters of this thesis.

6 Wavenets

The use of sigmoid functions in typical NNs might converge in a local minimum using a training algorithm. Trying to keep it simple, instead of a rough feedforward NN, wavenets are a variant of them and solve the problem of local minimum finding always the global minimum solution.

A wavenet (WN) is a generalization of a radial base function network (RBFN). The RBFN are actually feedforward NN that use radial base functions, like wavelets, as activation functions instead of logistic functions, like sigmoids. Knowing this, the basic structure of the WN is just one hidden layer apart from the input and output layers.

The nodes, or so called wavelons of the WN, are vectors of wavelets, as activation functions, applied on the inputs. Every single wavelon is multiplied to a weigh, which corresponds to the coefficients of the function expansion seen in the expression (4.6), also known as the wavelet base to a certain subspace V_j. Having that said, if a WN learns how to approximate any function following the expression (4.6), the WN is always applying Mallat's multiresolution analysis to do so.

By using radial base functions, the result only depends on the distance from the origin of one variable. Computing the value of the input at a single point in the wavelon or updating the estimated function from a new local measure involves only a small subset of coefficients [7, 11], thanks to the orthogonality property, if not, every single time the estimated function is updated all the coefficients should be calculated at once.

The output, following a series expansion, is generally expressed by the equation (4.6). This is why the hidden layer grows in each update of the estimated function, and every update corresponds to a level in the sum from the same expression, determined by the level of resolution from the Mallat's multiresolution analysis.

WNs also provide information for the relative participation of each wavelon to the function approximation and the estimated dynamics of the generating process.

6.1 Unidimensional function prediction

In this part of the project's chapter, it will be exposed the simplest system for a WN. Knowing that the number of mother wavelets is computed as $2^E - 1$, where E corresponds to the number of inputs given to the WN, it is possible to know the number of mother wavelets, or simply called wavelets, that perform the orthogonal base with a single scale function. For one input, the predicted function depends only on one dimension, and can be expressed by the equation (4.6), particularized in the form seen in (6.1).

$$f(x) = d\phi(x) + \sum_{m=0}^{M} \sum_{n=0}^{N} c_{m,n}\psi_{m,n}(x) \tag{6.1}$$

Where, the new parameters seen in the expression mean:

M: maximum level of resolution.

N: maximum translation of the rank 2^m wavelet where its value is not null.

Using the mother wavelet equation (5.2), with the defined values a_0 and b_0, the mother wavelet

ETSEIB

can be seen in (6.2).

$$\psi_{m,n}(x) = 2^{m/2}\psi(2^m x - n) \tag{6.2}$$

Parameter n will take values depending on the rank of the wavelet, from zero to N, where $N = M - 1$, with M being the rank, corresponding to 2^m for the resolution m.

The wavelet's input is an array of the different values from the variable that will be computed in the scale function and the wavelet. Defining the input variable as x, there will be i indices in the array, for $i = 1, 2...X$. With this notation, a model $\hat{y}$ can be expressed as a matrix system given by the following equation (6.3).

$$\hat{y}(x) = F(x)\sigma \tag{6.3}$$

The σ vector corresponds to the calculated weights for the WN, and finding the best model that predicts the $f(x)$ will depend on the solution for this vector. The generic dimensions of (6.3) as a matrix form are seen in table 6.1.

Table 6.1- Matrix dimensions.

Matrix	Dimension
$\hat{y}$	X x 1
F	X x J
σ	J x 1

Where J corresponds to the growth rate of the wavelons. $J = 1 + \sum_{m=0}^{M} 2^m$ in a unidimensional WN.

6.2 Multivariable function prediction

Here it will be exposed a bidimensional WN, to simplify the characterization, but once a wavelet base is defined and the growth rate of the WN is set, any multivariable function can be approximated without effort with enough computational power.

To know how many wavelets there are in the wavelet base, using the expression at the beginning of chapter 6.1, for a two-input case, it is set to three wavelets. The wavelet family will have the following form, seen in (6.4).

$$\psi^h_{m,n}(x_1, x_2) = 2^m\psi(2^m x_1 - n, 2^m x_2 - n) \qquad h = 1, 2, 3 \tag{6.4}$$

The 2^m coefficient multiplying the wavelets here, as the $2^{m/2}$ coefficient seen in (6.2) are meant to normalize the base, to make it orthonormal, but it does not matter for the goals of this project as long as the base is orthogonal. The coefficients on the σ vector will take the required value to approximate the given function and the norm will be implicit in that value.

The scale function will also be bidimensional, and using the same scale function for every input variable gives an advantage to simplify the calculus, as the scale function will have the expression in (6.5).

$$\phi(x_1, x_2) = \phi(x_1)\phi(x_2) \tag{6.5}$$

If the scale function is a combination of more than one scale function, here takes significance the first sum term in the expression (4.6), where now the function $\phi(x_1, x_2)$ has the following general expression (6.6).

$$\phi(x_1, x_2) = \sum_{j=1}^{N_\phi} \sum_{i=1}^{N_\phi} \phi_i(x_1)\phi_j(x_2) \tag{6.6}$$

Where:

N_ϕ: is the total number of scale functions.

And characterizing the expression for a two-functions case, the specific base expression results in a vector seen in (6.7).

$$\Phi(x_1, x_2) = [\phi_1(x_1)\phi_1(x_2), \phi_1(x_1)\phi_2(x_2), \phi_2(x_1)\phi_1(x_2), \phi_2(x_1)\phi_2(x_2)] \tag{6.7}$$

To make it simple, the following expressions will be focused in a single scale function case. Using the (6.5) expression, the mother wavelets from the wavelets' family seen in (6.4), are expressed as a base in (6.8).

$$\Psi(x_1, x_2) = [\phi(x_1)\psi(x_2), \psi(x_1)\phi(x_2), \psi(x_1)\psi(x_2)] \tag{6.8}$$

The growth rate of the WN, set by the number of wavelons needed in every resolution level is expressed as seen in the equation (6.9).

$$Wavelons = 1 + \sum_{m=0}^{M} (2 \cdot 2^m + 2^{2m}) \tag{6.9}$$

Finally, it can be generalized a formula, as a sum of terms, for the model $\hat{y}$ in a bidimensional WN, as seen in (6.10).

$$\begin{aligned} \hat{y}(x) = d\phi(x_1)\phi(x_2) + \sum_{m=0}^{M} (\sum_{i=1}^{N'} \sum_{n=0}^{N} c_{m,i}\phi(x_1)\psi_{m,n}(x_2) + \\ + \sum_{i=N'+1}^{2N'} \sum_{n=0}^{N} c_{m,i}\phi(x_2)\psi_{m,n}(x_1) + \\ + \sum_{i=2N'+1}^{2N'+N'^2} \sum_{l=0}^{N} \sum_{k=0}^{N} c_{m,i}\psi_{m,l}(x_1)\psi_{m,k}(x_2)) \end{aligned} \tag{6.10}$$

Where $N' = 2^m$. The matrix system will follow table 6.1 taking into account the wavelons' growth changes seen in the formula (6.9) and the development seen in (6.10).

6.3 Linear relations and offset parameter

In a WN, the output is established as a sum of linear relations among the values of the wavelons, and these linear coefficients are determined by the weights located between the hidden layer and the output layer, as explained in previous chapters. It can be introduced two new types of weights that are not treated as the weights associated to each wavelon.

The first type of weight corresponds to a direct linear relation between any input and the output. To apply this relation in the WN, the normalized data going into the wavelons is multiplied directly to a weight and the result is summed to the predicted output, it is done by adding a column of normalized inputs in the wavelet matrix $F(x)$.

The second type of weight is just a constant value, called offset, applied to the output. There is the option of getting an independent weight to the output by adding a column of ones in the wavelet matrix $F(x)$, and then multiplying the vector to a weight. It can also be used a *haar* scale function, seen in table 4-1, with the normalized inputs, the *haar* will take value 1 and the weight will be calculated.

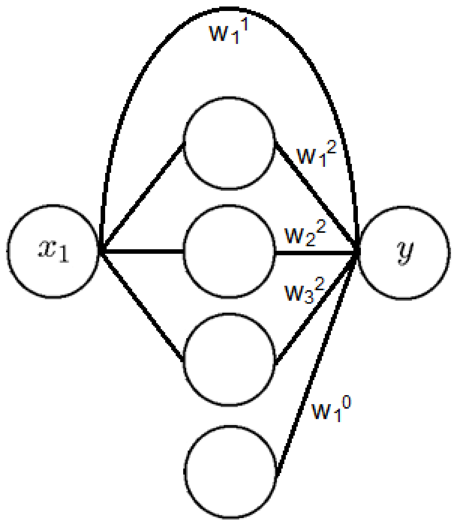

Figure 6-1. Linear term represented as the weight w_1^1 and the offset term as the weight w_1^0.

The graphic representation of the linear relation and the offset is shown in figure 6-1, following a similar WN structure found in [11].

6.4 Exponential wavenet's growth

The main clue in regards to building a WN is to know exactly its growth in every resolution's level. As seen in J from table 6.1 and the expression (6.9), the WN's growth is a linear relation among a set of exponentials determined by the number of inputs the WN has. Empirically, it can be found a general expression for the variation of WN's wavelons, according to the number of inputs and the level of resolution, as seen in (6.11).

$$Wavelons = 1 + \sum_{m=0}^{M} \sum_{i=1}^{E} K_i 2^{im} \tag{6.11}$$

Where:

M: maximum level of resolution.

E: number of inputs

K: multiplicity constant that represents the number of wavelets from the same mother wavelet for each i input, defined in the second level of resolution.

The expression is followed by three rules:

1st rule: $K = 1$ if $i = E$

2nd rule: $K_{i_1} = K_{i_2}$ where $i_1 + i_2 = E$

3rd rule: $\sum_{i=1}^{E} K_i = 2^E - 1$

It can be defined a generic second level of resolution, corresponding to the first orthogonal base, to easily find all K constants from (6.11) thanks to these rule,s and extrapolate the WN's growth through each level doing a minimum amount of operations. Doing so, it is possible to create table 6.2 rapidly.

Table 6.2 – WN's growth according to different number of inputs.

Inputs	ΔWavelons
1	$1 \cdot 2^{1m}$
2	$1 \cdot 2^{2m} + 2 \cdot 2^{1m}$
3	$1 \cdot 2^{3m} + 3 \cdot 2^{2m} + 3 \cdot 2^{1m}$
4	$1 \cdot 2^{4m} + 4 \cdot 2^{3m} + 6 \cdot 2^{2m} + 4 \cdot 2^{1m}$
5	$1 \cdot 2^{5m} + 5 \cdot 2^{4m} + 10 \cdot 2^{3m} + 10 \cdot 2^{2m} + 5 \cdot 2^{1m}$
6	$1 \cdot 2^{6m} + 6 \cdot 2^{5m} + 15 \cdot 2^{4m} + 20 \cdot 2^{3m} + 15 \cdot 2^{2m} + 6 \cdot 2^{1m}$

The independent term in (6.11) corresponds to the single wavelon with the scale function as the activation function, then it is added any of the rows from the table above inside the summation with the maximum level of resolution as the upped bound of summation.

From a computational perspective, it is important to know how it behaves the exponentials' set in a WN. In the developed algorithm, every exponential corresponds to a for loop and every K constant matches the number of new variables created, in each one of these loops, to save the new vectors which will be added to the wavelet matrix $F(x)$.

That is, with the algorithm used to create the WN, if it is necessary to modify the number of inputs, it will be no hard work and the major part to be changed in the code is defined by the WN's growth behaviour expressed in (6.11) and the rules it follows.

7 Loss function. Mean squared error

Once the WN's structure is set, it has to be trained to obtain the best solution for the weights. With the matrix system obtained from the WN, similar to the one in (6.3). The mean squared error (MSE), as an objective function, will measure the quality of the predicted model $\hat{y}$ and it is used for regression tasks, like linear regressions. This loss will calculate the squared differences between the target and the model. Therefore, applying the mean, the more data is added the less error is yield.

The form of the MSE for matrix systems is a multiple linear regression expressed in (7.1).

$$MSE = \frac{1}{N}\left\|y(x) - \hat{y}(x)\right\|^2 = \frac{1}{N}\left\|y(x) - F(x)\sigma\right\|^2 \tag{7.1}$$

If the previous expression is developed it becomes (7.2).

$$MSE = \frac{1}{N}(y(x)^\intercal y(x) - 2y(x)^\intercal F(x)\sigma + \sigma^\intercal F(x)^\intercal F(x)\sigma) \tag{7.2}$$

As the MSE is the function to minimize to find the optimal solution for σ, the optimal condition will be $\dfrac{\partial MSE}{\partial \sigma} = 0$. Applying the product transposition property from matrices, the expression to solve is (7.3).

$$\frac{\partial MSE}{\partial \sigma} = -2y(x)^\intercal F(x) + \sigma^\intercal F(x)^\intercal F(x) \tag{7.3}$$

Which can be finally expressed as (7.4).

$$F(x)^\intercal y(x) = F(x)^\intercal F(x)\sigma \tag{7.4}$$

Being a new system in the $Ax = B$ form, where A is the covariance matrix $F(x)^\intercal F(x)$ and B the product of the transposed wavelet matrix $F(x)^\intercal$ by the target matrix y. As it is shown in (7.4), the formula is the same as the one used to solve the system (6.3) by least squares and it could have been the way to go since the beginning because it is a low rank system, as the wavelet matrix will have more rows than columns.

Instead of using least squares, it is often used the backpropagation algorithm in a feedforward NN to minimize the loss function, but it can converge to a local minimum, whereas the least squares solution is a global minimum.

7.1 Mean squared error regularizer

Following the studies conducted in [10], it is introduced a new term, a regularizer, in the objective function and as a result it becomes a regularized objective function. The meaning of the regularizer is to be able to smooth the curvature of the model's surface by reducing the oscillations around the data used as a target for the model, providing a great scalability.

The regularizer also optimizes the computational calculus avoiding the possible problems related to the inverse covariance matrix when solving the system in (7.4) to find the weight's vector. The maximum and minimum eigenvalues from the covariance matrix tend to zero and this drives an uncontrolled growth of the number of conditions given by the expression $\frac{\lambda^{max}}{\lambda^{min}}$. If the λ^{min} tends to zero the estimated weight's vector can be inaccurate due to the inverse.

The simplest regularizer is the norm vector of the squared weights $\|\sigma\|^2$, giving the new loss function (7.5).

$$G = \|y(x) - F(x)\sigma\|^2 + \gamma\|\sigma\|^2 \tag{7.5}$$

Where:

G: is the new loss function

γ: is a multiplier

Following the same methodology as with the MSE, the new optimal condition is $\dfrac{\partial G}{\partial \sigma} = 0$. The new expression to solve the weight vector is shown in (7.6).

$$F(x)^{\intercal} y(x) = (F(x)^{\intercal} F(x) + \gamma I)\sigma \tag{7.6}$$

With the (7.6) formula, it can be observed the $Ax = B$ form as well, but the single difference from a non-regularized loss function is the identity of multipliers in the A matrix, added to the covariance matrix. This way, it can be said that the equation in (7.6) is solved by least squares.

The expression of γ is given by (7.7).

$$\gamma = \mu\lambda^{max} \tag{7.7}$$

Where:

λ^{max}: corresponds to the maximum eigenvalue from the covariance matrix.

μ: is a constant to be found.

The value μ depends on the application and, as it is said in [10], the numeric stability from an inverse of a matrix is defined by the number of conditions related to the constant with the expression $\frac{1}{\mu}$, so $\mu = \frac{\lambda^{min}}{\lambda^{max}}$. For the FHN model it will be used the value $\mu = 1e - 18$, which has given the best results.

8 FitzHugh-Nagumo model prediction

The main objective of this project, as it is already said, is to be able to train a NN to perform as a real neuron. There are many models that can describe the behaviour of a neuron, as an excitable system, such as Morris-Lecar, Hodgkin–Huxley, Hindmarsh–Rose or Fitzhugh-Nagumo, being the last one the model used.

The ordinary differential equations that define the FHN model are the following:

$$\begin{aligned}\dot{y} &= -y(y-1)(y-a) + w + u \\ \dot{w} &= \alpha y + \beta w\end{aligned} \tag{8.1}$$

Where *y* and *w* are the state variables and *u* the input variable also known as forced term. The *u* variable is later referred as *Iapp*, and it represents the inhibition and excitation relations between the specific neuron and its neighbours.

Solving the system using Euler's method, for a given step size *h*, the resulting equations are:

$$\begin{aligned}y_{n+1} &= y_n + h(-y_n(y_n-1)(y_n-a) + w_n + Iapp) \\ w_{n+1} &= w_n + (\alpha y_n + \beta w_n)\end{aligned} \tag{8.2}$$

In order to train the system appropriately, the input *Iapp* will remain constant enough time so that the system reaches a stable point or a stable orbit, therefore the system has enough time to reach these two stable configurations and will not waste time in an already explored state, that is, the *Iapp* changes to explore the new state as soon as the previous one is totally stable.

8.1 Three inputs case

To be able to predict the FHN model, the WN has to get an initial condition composed by any *y* and *w* and then output the next *y* and *w* point in the same *Iapp* input, as if the system was solved using Euler's method. With an initial y_n and w_n, it will calculate the y_{n+1} and w_{n+1} of the same *Iapp*. Then the WN will be able to predict any y_{n+1} and w_{n+1} from any given *Iapp* with just one initial condition. This idea is represented with the WN's structure shown in figure 8-1, seen below.

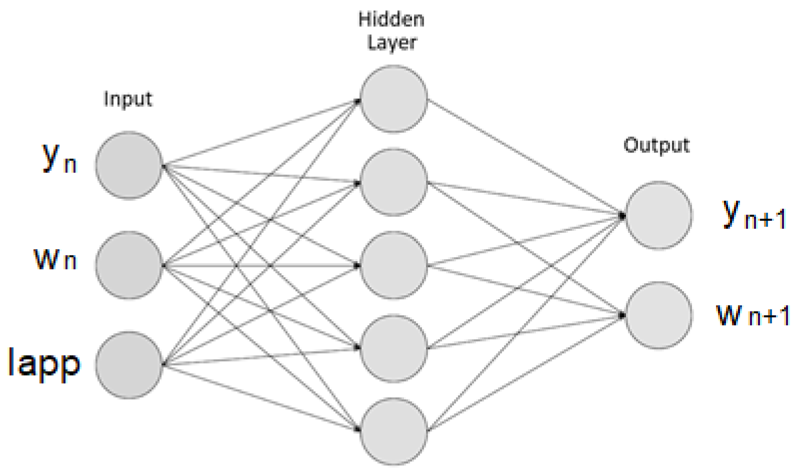

Figure 8-1. WN's structure for the FHN model.

To achieve this, the WN has to be trained with random *Iapps* and one period delay outputs, as inputs, to be able to calculate the weights' vector.

The reason why it is necessary to randomise the *Iapps* used to train the WN is because it has to learn from different transient states, to comprise all possible responses from how the system is externally excited. And the reason why the two other inputs are a one period delay outputs is a way to stick to the Euler's method nature, for solving the system and generating data to create a target the WN can predict.

To approximate the FHN model through an entire grid of points in every *Iapp* from any possible initial grid's point, the WN has to explore a considerable range for the variables y, w and *Iapp*, visualising it as a 3D space.

To do it so, there are two possible ways. The first one consists in defining every single point of the 3D space with the coordinates w, y and *Iapp* as inputs and then calculate the next integration step of every one of them using Euler and using the result as a target for the output. This first method is computationally intense, on that account a second way is thought. This second one consists in one single initial condition used to solve a given number of integration steps from the Euler's method in a single *Iapp* and then change to the next *Iapp* using the last y and w result as the new initial condition in the following *Iapp* and repeating the process a given number of different *Iapps*.

Finally, the wavelets used in the WN are the *hat*, the *quadratic spline* and the *bicubic spline*, seen in table 4-1.

The *hat* is simple and can be used for fast results but the C^1 class function that defines it can make it hard for a first prediction if the WN does not make a good solution for the weights' vector. Whereas the *quadratic* and the *bicubic splines* have a non-null second derivative and they have smoother solutions to every next resolution level. As the FHN model is non-linear, it might be a good option to work with spline wavelets. The *quadratic* should be enough for a good prediction but for the sake of fun, the *bicubic* is also proposed to see if it can have other benefits over the *quadratic spline*.

8.2 Activation function superposition

Since now, the WN's growth is expressed as a result of a single activation function defined in the domain of the scale function used. Using a single activation function makes the WN less capable to approximate a function with high accuracy rather than using the same activation function superposed a number of times in the same domain, due to the fact that the extremes of de domain the scale function takes values near zero and these values define the wavelet matrix.

As it is studied in [10], there are a series of advantages by using superposed functions, such as the need for less resolution levels for an accurate result and an accurate approximated target using less sampled data.

In this project it is studied the impact of the number of superposed scale functions using a *hat*, a *quadratic spline* and a *bicubic spline*. In figure 8-2 it can be seen the approach used for the superposition using a *quadratic spline* as an example. There is always a single function that exists through all the domain and then the same function is added in symmetrical translations.

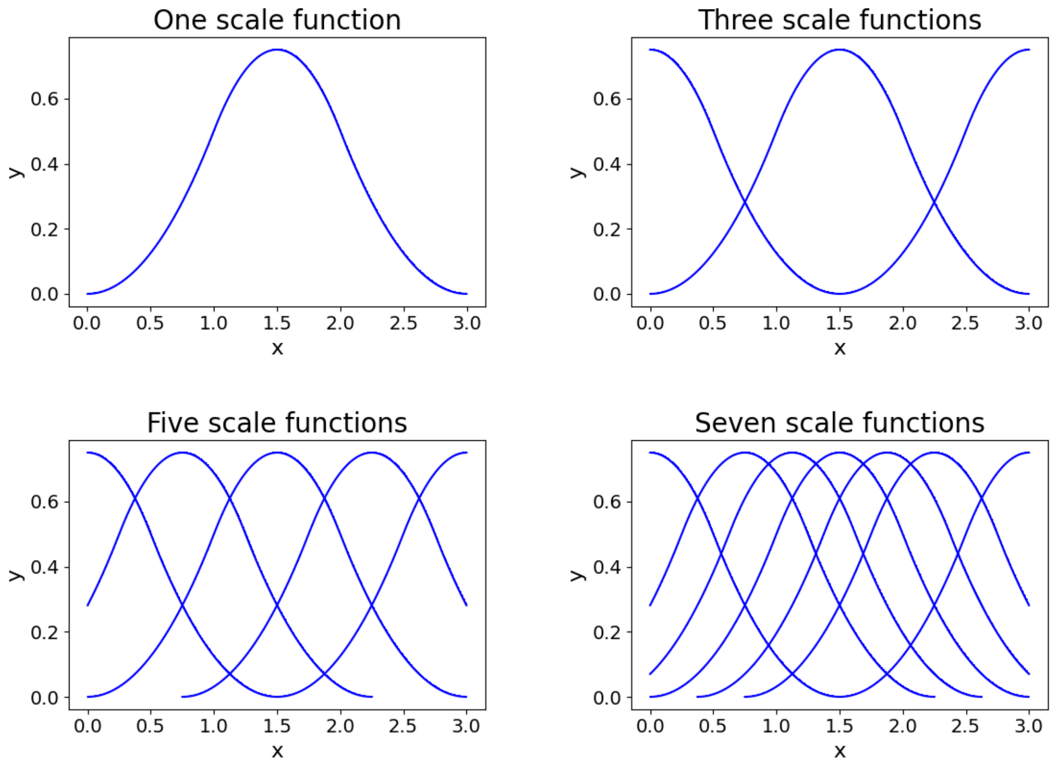

Figure 8-2. Superposed quadratic spline scale functions.

Looking at figure 8-2, it can be understood that a weight multiplied by a small value in the wavelet matrix $F(x)$ will have a small impact compared to the weights associated to values near the centre of the domain, obtaining worse results for the approximated target function.

As wavelets are expressed as a sum of scale functions, as seen in (5.4), the superposition property is also applied in wavelets by just superposing the scale functions. In the second level of resolution and further the activation function will be a superposition of wavelets.

With this final approach the WN's growth has to be redefined, using the expression in (6.11), as shown in the equation (8.3).

$$\Delta Wavelons' = J^E(1 + \Delta Wavelons) \tag{8.3}$$

Where:

J: is the number of superposed functions.

E: is the number of the WN's inputs.

$\Delta Wavelons$: is the wavelons' increment, already defined in the chapter 6.4.

$\Delta Wavelons'$: is the new wavelons' increment.

8.3 Optimal wavelons' configuration

In this chapter it will be discussed the optimal configuration for the WN. It will depend on three main parameters: the scale function used, the number of superposed functions and the level of resolution. The best configuration will be decided with the relative mean squared error (RMSE), defined by the expression (8.4).

$$RMSE = \frac{\sum_{i=0}^{N}(y_i(x) - \hat{y}_i(x))^2}{\sum_{i=0}^{N}(y_i(x) - \overline{y})^2} \tag{8.4}$$

Where:

$\hat{y}$: is the predicted output

$\overline{y}$: is the target's mean

y: is the target

It is quite obvious that as more resolution is computed the less error will be yield, and the same happens with the number of superposed functions. As for the scale function used, the *bicubic spline* should obtain better results as it is a smoother function than the *quadratic spline*, and this in turn has smoother variations per resolution rather than the *hat*, as it was already mentioned in the chapter 8.1.

Figure 8-3 shows the results of twelve different configurations. All trained with the same inputs, the same initial condition, and the same new condition used for predicting the new target, after the target function approximation was done.

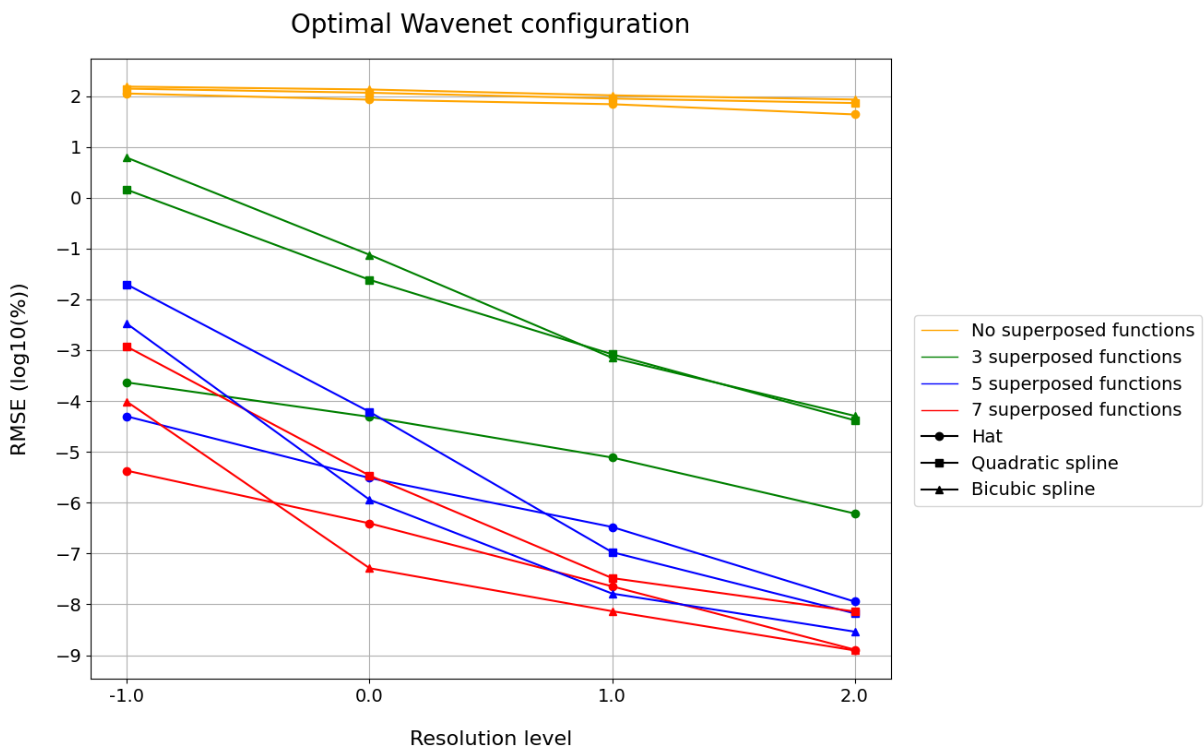

Figure 8.3. Optimal Wavenet configuration from an error-resolution perspective.

As it was thought, the best error is achieved using the fourth level of resolution (the highest one simulated), for an $m = 2$ in the WN's growth, using a *bicubic spline* and the maximum number of superposed scale functions set in the simulation, seven. The MSE reached is $3,08 \cdot 1e-13$ and the RMSE is $1,24 \cdot 1e-9$. Despite what was thought, the *hat* shows also a really good behaviour among the different configurations. And it can be seen that the curves tend to decrease the error as the level of resolution is increased.

The error may be the most important factor to determine if a model is good enough but the time needed to train it may be a concern. Time depends in a huge amount of variable, going from the hardware of the computer used for the simulation such as the CPU architecture, the CPU clock speed, the RAM capacity, the GPU if it is used, to the code itself such as the language used or the programmer's capabilities.

In figure 8-4 it can be seen the same number of different WN's configurations as before but taking in consideration the time spent to train the WN in each case. As it is mentioned, time can drastically vary depending on the machine or the algorithm used to create the WN, but from a qualitative point of view it can be seen that the error decreases when time increases.

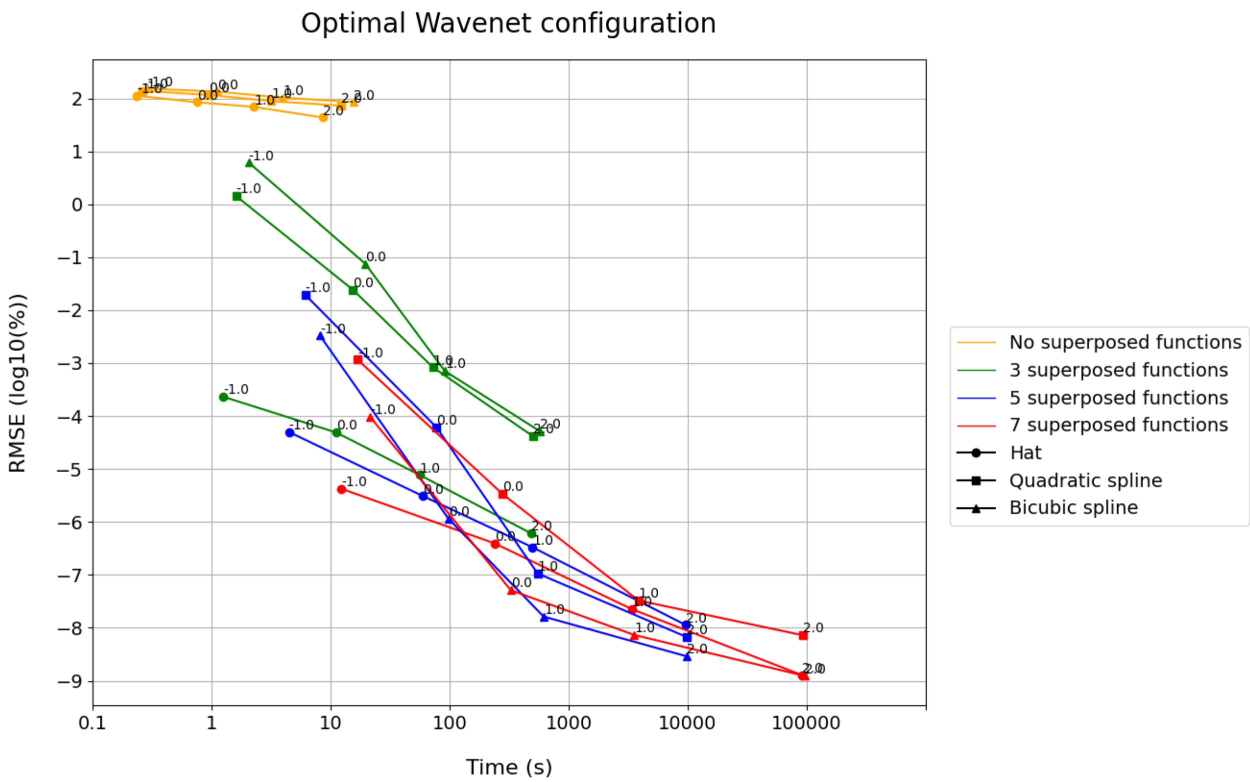

Figure 8-4. Optimal Wavenet configuration from an error-time perspective.

It will be left to one's preferences to consider what is better for each case, if the error is more important than time, or both need to be considered. As this kind of results will depend on the neuron model used and the algorithm to solve the WN's loss function, it might be a good idea to execute this kind of graphics every time a particular WN has to be studied to see a good balance between the error willing to assume to speed up the studies.

As the configuration of five superposed functions and a fourth level of resolution has a considerable low relative error among other simulations and the time spent in each simulation are not more than three hours, whereas the seven superposed functions case with the same resolution level has slightly more than a full day simulation time, the five scale functions case is chosen for further studies with the WN.

8.4 Effect of the linear terms in the error

Once a WN's configuration is set, it is time to see if it is useful to add a linear term as exposed in [11]. To see the effect of linear terms it has been simulated three versions of a WN. The first with only two wavelons, with $f(x) = x$ as the activation function, in the first level being each one a linear term for w and y, the second with only scale functions in the first level of resolution and the third one being a combinations of scale functions and the linear terms in the first level. The rest of the resolution levels will include the wavelons with the corresponding wavelets in the WN.

The reason why it is created the first of the three mentioned WN versions, is to know if getting rid of the scale functions, in the first level, will worsen the approximation of the targeted function in later resolution levels, and make it harder for the WN to learn. Without the wavelons form the scale functions, in later levels the wavelet base will contain fewer elements and the WN will have to approximate the same target with a lower resolution base.

The final results of the three WN versions are shown in figure 8-5.

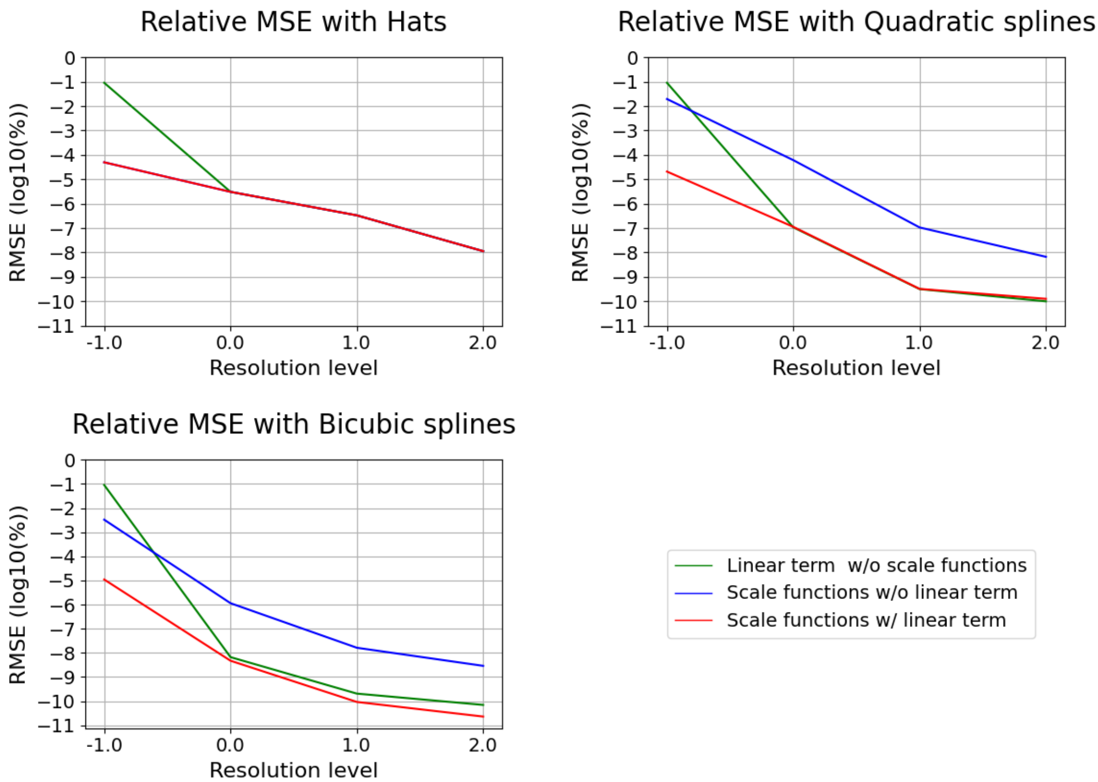

Figure 8-5. Effect of the linear terms in the error.

It can be seen that, at least in this particular FHN model, the linear terms do affect reducing the error by 100 times in the cases of the *quadratic spline* and the *bicubic spline*, as seen by the difference represented by the blue and the red lines in figure 8-5.

In every case it can be appreciated the importance of the wavelets from the second level of resolution, where they can approximate the targeted function without the need of the scale functions' wavelons of the first level, as represented with the green line in figure 8-5. If a low error WN is needed, it can be defined without the scale functions in the first level, and use the linear terms along with the wavelets, reducing the number of wavelons, thus reducing the training time.

In the *hat* case the blue and red lines are equal, this means that the use of a linear term with the scale functions does not affect in the error over the case with no linear term. It may be due to the properties of the *hat* function, as it is a linear function with two gradients. The relative participation in the function approximation of the linear term is already fulfilled by the *hat* itself, and the WN learns that there is no need for the linear terms.

In a first place it was intended to be used an offset parameter, as explained in chapter 6.3. It was proposed thinking about the *Iapp* variable, as the only plausible offset, looking the FHN model in (8.1). Considering it as a constant, was only useful for a single *Iapp* case training. If there is a will to explore multiple *Iapps* to train the WN through various transient states to comprise all possible responses, it would have been needed to train a new WN or every single different *Iapp*. It was non-viable, and the idea of an offset parameter was discarded.

8.5 Final results

In figure 8-6 there are displayed the results of the final WN approximating the FHN model with a *bicubic spline*, five superposed scale functions and a fourth level of resolution and then predicting the outputs from a different initial condition.

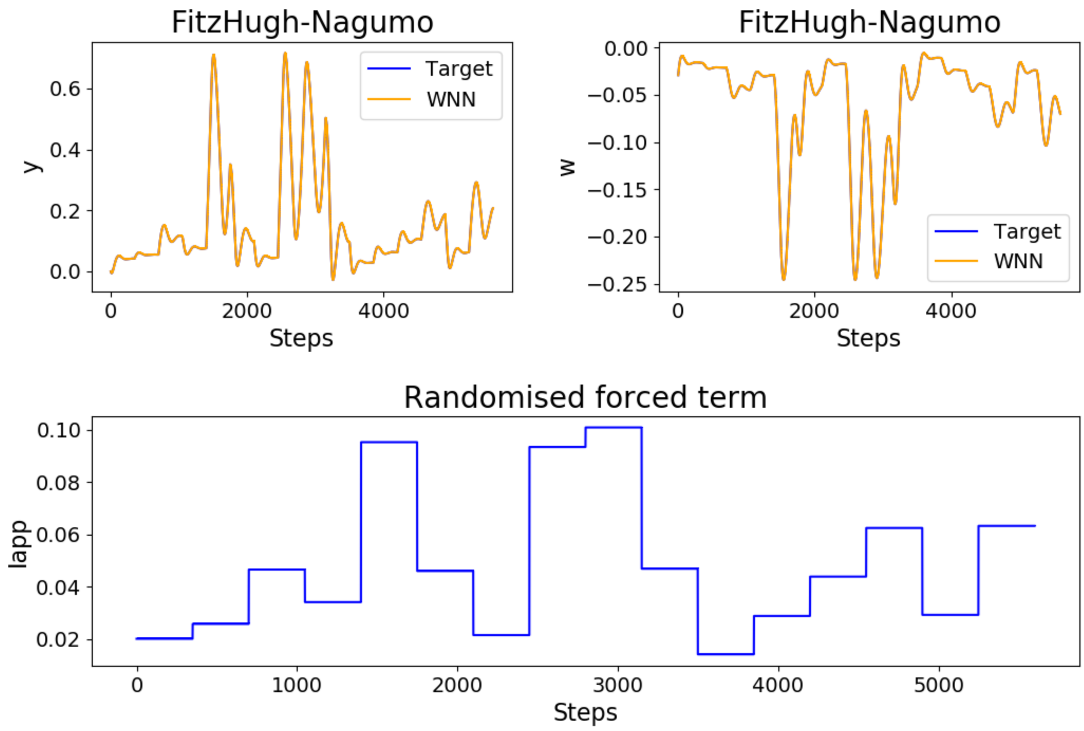

Figure 8-6. Simulation results.

As it can be intuited from the error shown in figure 8-5, the WN has reproduced the y and w outputs, drawing the orange line over the blue one.

In figure 8-7 there is displayed the 3D phase portrait of the FHN model within the same conditions as the results shown in figure 8-6. As it can be seen, the WN has reproduced exactly the same diagram over the target.

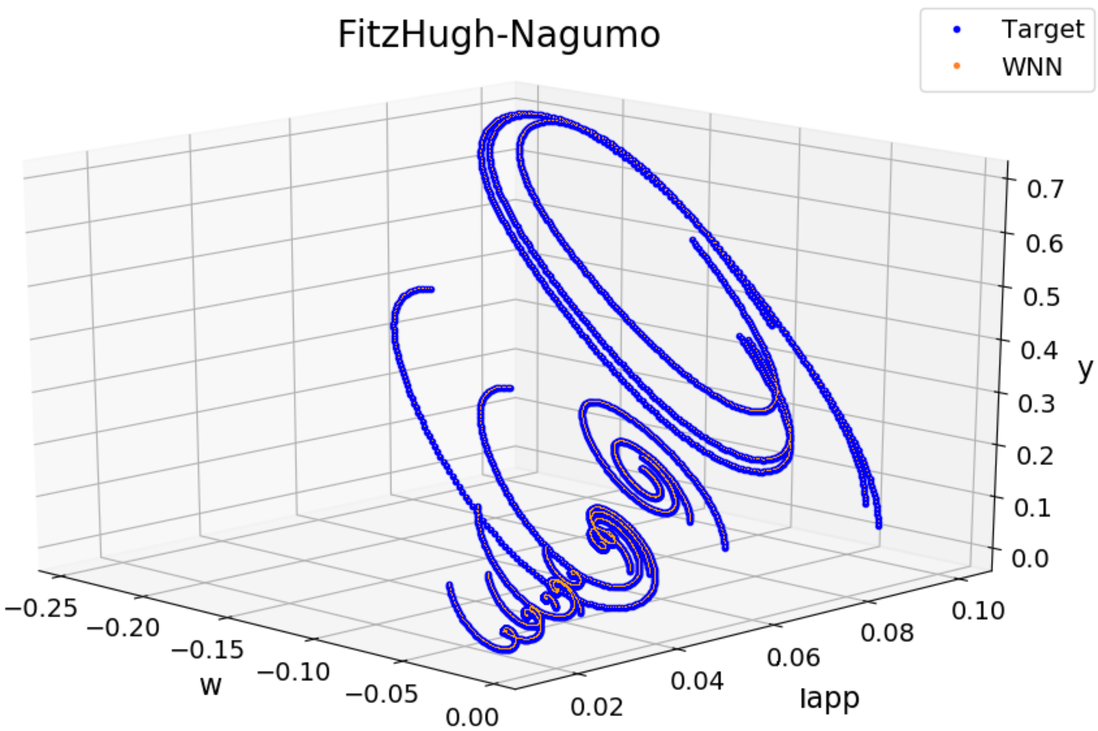

Figure 8-7. Phase portrait of the FHN model.

9 Economic balance

In this chapter it is computed the costs associated to the development of the project taking into account human resources, the engineering such as modelling and development of the WN, any licence if used and the electricity bill.

The number of hours is charged with a 50€/h price. Simulation is also considered as important as any other engineering part due to the fact that while simulations are running it cannot be changed and tested anything about the current model.

In this project no licence is used, as the software needed it is been Ubuntu 19.10, a free Linux distribution, and emacs as a text editor for Python 3.7 coding. The report writing has been done in LaTeX, a free text editor widely used for papers and many types of publications and no paid Python library was needed.

Finally, the electricity bill is added to the total cost, not only counting the hours spent by human resources and engineering but also considering the hours spent with the various simulations done with the WN. The computed cost is shown in table 9.1, with a total cost of 20755,73€.

Table 9.1. Economic balance of the project.

Concept		Hours	Cost [€]
Research and study		50,00	2500,00
Modelling and development		200,00	10000,00
Simulation		120,00	6000,00
Report writing		30,00	1500,00
Meetings		15,00	750,00
Concept	**Units [kWh]**	**Price/unit [€/unit]**	**Cost [€]**
Electricity	51,875	0,1105 €/kWh	5,73
Total cost			**20755,73**

The consumption considered for the electricity bill is computed with the consumption under average load of the computer used during the project [12] and the mean price per kWh in a year in Spain [13].

10 Environmental Impact

The environmental indicator that reflects the impact of the greenhouse gases emitted directly or indirectly, corresponds to the carbon footprint, which is measured in the equivalent CO_2 mass.

To know the range of the footprint it is going to be taken in consideration the indirect emission due to the consumed electricity to fulfil the project. Following the methodology described in PAS 2050:2011 [14] by the British Standard Institution (BSI) Group, who define the specifications to evaluate the greenhouse gases emissions, it is used the factor of 0,385 $kgCO_2eq/kWh$.

Using the kWh defined in table 9.1 to indicate the consumption during the project and the factor provided by the BSI, it can be found the total mass of CO_2 emitted in the atmosphere during the realization of this project, shown in table 10.1.

Table 10.1. Carbon footprint of the project.

Concept	Units [kWh]	Kg CO_2eq/kWh	Kg CO_2eq
Electricity	51,875	0,385	19,972

11 Conclusions

To sum up, the final results show that it is been possible to create a working WN based on the FHN model, it has been able to predict the behaviour of a black box excitable system, like one described by a neuron, following the studies conducted in [3].

All the objectives have been achieved. As it has been exposed, the concepts about the theory of wavelets have been understood deeply enough to use them to define an algorithm useful for the WN's purpose. Developing the empirical growth rate for the wavelons, based on the number of inputs, has given a simplistic point of view for writing the algorithm's code around it.

Many WN's structures have been studied to be able to know which one is suited for a desired scenario, as shown in chapter 8.3. The error-time graphic can help to decide how much error can be sacrificed in favour of a reduction in time, useful for future studies and development computationally intensive. Studying the implementation of a linear term, as seen in chapter 8.4, has also been positive for achieving a slightly reduction in training time and a significant reduction in error, for non-linear scale functions' implementation in the WN.

Notwithstanding the three months spent in this project, it has been nearly impossible try to implement a parallelized work flow to increase the computational performance, nonetheless is was improved the by using the minimum number of variables and preallocated variables to reduce the work done by the memory.

As a final thought for future work, it can be a good idea to explore the use of a second neural network to work beside the WN. The second NN could be responsible for finding the best frequency for the Iapp to make the system reach, as fast as possible, to stable point with less oscillations, whereas for an Iapp in a stable orbit could have a fixed frequency. And there will be no need to fix a small frequency, enough to explore every possible system response as it is already done.

12 References

[1] T. P. Trappenberg, "Fundamentals of Computational Neuroscience," 2010.

[2] M. S. Keil, E. Roca-Moreno, and Á. Rodriguez-Vázquez, "A neural model of the locust visual system for detection of object approaches with real-world scenes," *Proceedings of the Fourth IASTED International Conference on Visualization, Imaging, and Image Processing*, no. February, pp. 340–345, 2004.

[3] N. Roqueiro, C. Claumann, A. Guillamon, and E. Fossas, "A Black-box Model for Neurons," *2019 IEEE 10th Latin American Symposium on Circuits and Systems, LASCAS 2019 - Proceedings*, pp. 129–132, 2019.

[4] M. Bergman, "Containing the sun," Harvard, p. 1, 2019. [Online]. Available: https://news.harvard.edu/gazette/story/2019/04/harvard-princeton-scientists-make-ai-breakthrough-for-fusion-energy/

[5] A. Tch, "The mostly complete chart of Neural Networks, explained," 2017. [Online]. Available: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

[6] S. Verma, "Understanding different Loss Functions for Neural Networks," 2019. [Online]. Available: https://towardsdatascience.com/understanding-different-loss-functions-for-neural-networks-dd1ed0274718

[7] G. Mallat, "A Theory for Multiresolution Signal Decomposition : The Wavelet Representation," vol. II, no. 7, 1989.

[8] A. D. Señales, "Introducción a la Transformada Wavelet," *... de Señales y sistemas. Universidad de Navarra*, 2006. [Online]. Available: http://www.exa.unicen.edu.ar/escuelapav/cursos/wavelets/apunte.pdf

[9] S. T. Ali, J.-P. Antoine, and J.-P. Gazeau, "Multidimensional Wavelets," pp. 307–330, 2000.

[10] C. A. Claumann, "Desenvolvimento E Aplicações De Redes Neurais Wavelets E Da Teoria De Regularização Na Modelagem," p. 167, 2003.

[11] A. K. Alexandridis and A. D. Zapranis, "Wavelet Neural Networks : A Practical Guide," 2006.

[12] F. Glaser, "Análisis completo del Asus ROG Strix GL702ZC (Ryzen 7 1700, Radeon RX 580)," p. 1, 2017. [Online]. Available: https://www.notebookcheck.org/Analisis-completo-del-Asus-ROG-Strix-GL702ZC-Ryzen-7-1700-Radeon-RX-580.248850.0.html

[13] Tarifasgasluz, "¿Cuánto cuesta la luz al mes?" p. 1, 2020. [Online]. Available: https://tarifasgasluz.com/faq/cuanto-cuesta-luz-mes{#}:{~}:text=Elpreciomediomensualdel,propiosdescuentos{%}2Cpromocionesycondiciones.

[14] BSI, "PAS 2050:2011 Specification for the assessment of the life cycle greenhouse gas emissions of goods and services. British Standards Institution, London," pp. 1–45, 2011.